Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Rok Koleša

Iskanje ničel polinoma z vstavljanjem vozlov

diplomsko delo

univerzitetni interdisciplinarni študijski program prve stopnje računalništvo in matematika

Mentor: prof. dr. Gašper Jaklič

Ljubljana 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.
Slike so narejene s pomočjo orodja Mathematica.

Fakulteta za matematiko in fiziko ter Fakulteta za računalništvo in informatiko izdajata naslednjo nalogo:

Tematika naloge:

V diplomskem delu obravnavajte metodo Mørkena in Reimersa iz leta 2007 za iskanje ničel polinomov in zlepkov. Metoda polinom najprej predstavi v obliki B-zlepka, nato pa izkoristi dejstvo, da kontrolni poligon zlepka dobro aproksimira zlepek. Z vstavljanjem ničel kontrolnega poligona v seznam vozlov zlepka dosežemo konvergenco k ničli zlepka. Metoda je hitra in stabilna. V delu opišite metodo in jo implementirajte v primernem programskem jeziku. Preizkusite jo na primerih.
Osnovna literatura: K. Mørken, M. Reimers: An unconditionally convergent method for computing zeros of splines and polynomials.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Rok Koleša, z vpisno številko **63110263**, sem avtor diplomskega dela z naslovom:

Iskanje ničel polinoma z vstavljanjem vozlov

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Gašperja Jakliča,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 17. septembra 2014 Podpis avtorja:

Kazalo

Povzetek

V tem diplomskem delu opišemo nov način iskanja ničel polinoma. Najprej spoznamo interpolacijo, odsekoma polinomske funkcije in B-zlepke. Polinom predstavimo v obliki B-zlepka, potem pa izkoristimo tesno povezanost polinoma z njegovim kontrolnim poligonom. Glavna ideja je, da poiščemo neko ničlo poligona, kar nam služi kot prvi približek, jo vstavimo med vozle in modificiramo celoten poligon. Postopek ponavljamo, dokler približki ne konvergirajo. Vse ničle dobimo tako, da najprej poiščemo prvo ničlo, počistimo vse približke, ponastavimo vse spremenljivke in celoten postopek ponovimo od tiste točke naprej. Podana je tudi MATLAB implementacija algoritma.

Ključne besede: polinom, ničla, interpolacija, vozel, B-zlepek, MATLAB, poligon.

Abstract

In this thesis we present a new way of finding polynomial roots. First we find out what is interpolation, what are piecewise polynomials and what are B-splines. We write down the polynomial as a B-spline and then we use the close relation between the spline and it's control polygon by finding a root of the polygon, insert that root as a knot which we use as our first estimate and modify the polygon. The procedure is repeated until the estimates converge. We then proceed to find all the other roots by reseting all variables and start the whole procedure from the found root onwards. At the end we give a MATLAB implementation of the algorithm.

Keywords: polynomial, root, interpolation, knot, B-spline, MATLAB, polygon.

Poglavje 1

Uvod

Eden najbolj osnovnih in najpogostejših problemov v matematiki je zagotovo iskanje ničel funkcije. V tem diplomskem delu se bomo osredotočili na polinome. Za polinomske funkcije poznamo dobre metode, kot je na primer algoritem Jenkins-Traub [2], toda ubrali bomo nekoliko drugačen pristop. Spoznali bomo absolutno numerično stabilno metodo, ki sta jo leta 2007 razvila Norvežana Knut Mørken in Martin Reimers [5]. Temelji na interpolaciji funkcije oziroma polinoma z B-zlepki in na izkoriščanju tesne povezanosti med zlepkom in njegovim kontrolnim poligonom.

Kontrolni poligon je lomljenka, ki jo dobimo tako, da kontrolne točke eno za drugo povežemo med sabo z ravno črto. Tako dobimo zelo ohlapen približek polinoma, toda izkaže se, da je tudi ta približek izredno uporaben. Najbolj je pomembno dejstvo, da so ničle kontrolnega poligona skoraj ničle funkcije. Takoj ko najdemo eno tako ničlo, lahko rečemo, da je to prvi približek in na tistem mestu vstavimo še en vozel, s tem tudi kontrolno točko, in dobimo boljši poligon, nekoliko bolj podoben funkciji. Princip se tako ponavlja in ko določimo, da so ničle vse bolj skupaj oziroma da so konvergirale, postopek ustavimo. S tem smo poiskali najmanjšo, najbolj levo ničlo. Na tem mestu moramo približke počistiti in iskati nadaljnje ničle od pravkar pridobljene naprej. Če ne najdemo nobene ničle, pomeni da smo si izbrali napačen interval za iskanje, ali pa res ni nobene ničle. Kompleksne

rešitve so za nas nedosegljive.

Seveda pa je treba začeti na začetku. V nadaljevanju se bomo najprej posvetili polinomom, interpolaciji in zlepkom. Zgledovali se bomo po [4]. To teorijo bomo uporabili kot ogrodje za našo implementacijo, ki jo bomo na koncu tudi realizirali.

Algoritem je narejen po psevdokodi, ki jo lahko najdemo v [5], napisan pa je v programskem jeziku MATLAB.

Poglavje 2

Ničle polinomov

V splošnem lahko polinom ene spremenljivke stopnje n zapišemo kot

$$p(x) = \sum_{i=0}^{n} a_i x^i = a_n x^n + a_{n-1} x^{n-1} + ... + a_2 x^2 + a_1 x + a_0$$

kjer so a_i elementi kolobarja, nad katerim je polinom zgrajen, v našem primeru realnih števil.

Ničla je tako število x, da zadošča enačbi

$$p(x) = 0.$$

V nekaterih primerih je ničlo lahko poiskati, na primer za linearne in kvadratične polinome. Zaplete pa se, ko spremenljivka x nastopa v višjih potencah. Kot primer lahko vzamemo polinom

$$p(x) = x^4 + 6x^2 + 3x - 4, \tag{2.1}$$

kjer moramo rešitev poiskati ali numerično ali s Hornerjevim algoritmom ali pa s formulo. Ker Hornerjev algoritem temelji bolj na sreči in logičnemu premisleku, bomo kasneje ničle tega polinoma poiskali s pomočjo naše metode. Rešitvi enačbe

$$x^4 + 6x^2 + 3x - 4 = 0 \tag{2.2}$$

sta $x_1 = -1$ in $x_2 = 0.59185$. Obstajata še dve kompleksni rešitvi, saj je polinom stopnje 4. Ni pa nujno, da polinom sploh ima realno ničlo. Vedno pa ima vsaj eno kompleksno. To nam pravi osnovni izrek algebre.

Izrek 2.1 ***(osnovni izrek algebre)*** *Vsak polinom $p \in \mathbb{C}[x]$ stopnje vsaj 1 ima ničlo.*

V pomoč pri dokazu posledice tega izreka pa moramo najprej dokazati naslednjo trditev.

Trditev 2.2 *Naj bo p polinom stopnje vsaj 1 in $\alpha \in \mathcal{O}$ ničla polinoma p. Potem obstaja tak polinom $q \in \mathcal{O}[x]$, da je $p(x) = (x - \alpha)q(x)$.*

Dokaz. Polinom p lahko delimo s polinomom $(x - \alpha)$. Kot smo navajeni pri navadnem deljenju, lahko potem polinom p zapišemo kot

$$p(x) = q(x)(x - \alpha) + c, \tag{2.3}$$

kjer je $q(x)$ količnik, $(x - \alpha)$ delitelj in c ostanek, ki je tudi polinom stopnje manj kot 1. Vemo, da je α ničla polinoma p in vstavimo to vrednost v (2.3),

$$0 = p(\alpha) = q(\alpha)(\alpha - \alpha) + c = c. \tag{2.4}$$

Dobili smo, da je $c = 0$, torej je res $p(x) = q(x)(x - \alpha)$. □

Posledica 2.3 *Polinom $p \in \mathbb{C}[x]$ stopnje $n \geq 1$ ima n ničel, štetih z večkratnostmi.*

Dokaz. Vzemimo polinom p stopnje n. Po izreku 2.1 ima p ničlo α_1 in po trditvi 2.2 lahko polinom zapišemo kot

$$p(x) = (x - \alpha_1)q_1(x).$$

Postopek ponovimo, saj je stopnja polinoma $n - 1$ in če je $n - 1 \geq 1$, lahko polinom p zapišemo še kot

$$p(x) = (x - \alpha_1)(x - \alpha_2)q_2(x).$$

Postopek ponavljamo, vse dokler je stopnja na novo dobljenega polinoma enaka 0 in zapišemo:

$$p(x) = (x - \alpha_1)(x - \alpha_2) \cdots (x - \alpha_n)q_n(x).$$

Tu ima polinom $q_n(x)$ stopnjo 0, torej je konstanta, polinom p pa ima ničle $\alpha_1, \alpha_2, \ldots, \alpha_n$, teh pa je natanko n. □

Izreka 2.1 ne bomo dokazali, bralec pa lahko najde dokaz v [3], od koder sta povzeta tudi izrek 2.1 in trditev 2.2.

Sedaj lahko vidimo, da ima polinom vedno vsaj eno ničlo, kadar ga gledamo nad obsegom kompleksnih števil. Nas pa bolj kot kompleksne ničle zanimajo realne. Ker kompleksne ničle vedno nastopajo v konjugiranih parih, lahko iz tega sklepamo, da bodo polinomi lihe stopnje vedno imeli vsaj eno realno ničlo. Za polinome sode stopnje pa take reči ne moremo trditi. Naša metoda bo morala torej vedno vrniti nek rezultat, če bo polinom lihe stopnje. Če ga ne, bomo morali povečati interval. Pri polinomih sode stopnje pa se lahko zgodi, da ne obstaja realna rešitev. V tem primeru nam povečanje intervala ne koristi.

Poglavje 3

Interpolacija

3.1 Splošno o interpolaciji

V praktičnih primerih ponavadi nimamo danega predpisa za kakšno funkcijo ali polinom. Zanašati se moramo predvsem na opazovanja in opazovane vrednosti. Dane imamo samo pare $(x_i, f(x_i))$, ki jih je seveda končno.
Primer: V tabeli 3.1 so dane točke.

x	$f(x)$
0	0
1	1
2	4
3	9
4	16

Tabela 3.1: Tabela danih točk za interpolacijo.

Zanima nas, koliko bi bilo $f(1.5)$? V tem primeru lahko vidimo, da se tem petim točkam prilega funkcija $f(x) = x^2$ in lahko izračunamo $f(1.5) = 2.55$, toda v splošnem ne gre tako lahko.

Za interpolacijo se torej odločimo takrat, ko imamo na voljo le posamezne podatke o funkciji in jo želimo kar se da dobro aproksimirati. Seveda pa želimo interpolirano funkcijo $\tilde{f}$ zapisati v zaključeni obliki. V ta namen si moramo izbrati neko bazo.

Naj bodo $s_1, s_2, \ldots, s_n$ baza n-parametrične linearne družine. Povedano drugače, elementi $s \in S \subset X$, kjer je X unitaren vektorski prostor, oziroma za naše potrebe evklidski prostor, bodo oblike $s = \sum_{i=1}^{n} \alpha_i s_i$ za neke skalarje α_i. Interpolacijski problem je poiskati prav te skalarje tako, da za pripadajoči element s velja

$$\lambda_i s = r_i$$

za vsak $1 \leq i \leq n$, kjer so λ_i dani linearni funkcionali definirani nad $\mathcal{L}(\{s_1, s_2, \ldots, s_n\})$, r_i pa so znane vrednosti funkcije. Bolj enostaven, direkten način gledanja na interpolacijo pa je, da $\tilde{f} \in S \subset X$ interpolira $f \in X$, če za vse i velja

$$\lambda_i \tilde{f} = \lambda_i f.$$

Primeri najbolj pogostih linearnih funkcionalov so:

- vrednost f v točki x_i, torej $\lambda_i f := f(x_i)$
- povprečje f, torej $\lambda_i f := \frac{1}{\beta_i - \alpha_i} \int_{\alpha_i}^{\beta_i} f(x)dx$
- r-kratna vrednost f v točki x_i, torej $\lambda_i^l f := f^{(l)}(x_i), l = 0, 1, 2, \ldots, r-1$

Skalarji α_i niso nujno enolično določeni, celo ni nujen njihov obstoj. Če pa so enolično določeni za vsak nabor podatkov r_i, pa rečemo, da je interpolacijski problem korekten.

Če so $s_1, s_2, \ldots, s_n$ linearno neodvisni med seboj, sestavljajo bazo za prostor $S = \mathcal{L}(\{s_1, s_2, \ldots, s_n\})$. Primer baze so Lagrangeevi bazni polinomi

$$s_i(x) = \ell_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j},$$

ki se uporabljajo za polinomsko interpolacijo. Če podrobno pogledamo indekse, opazimo, da so ti bazni polinomi definirani od 0 naprej in že sami po sebi sestavljajo bazo za prostor P_n, torej polinomov stopnje $\leq n$. V prvem

poglavju smo uporabili potenčno bazo

$$P_n = \mathcal{L}(\{x_i\}_{i=0}^n),$$

poznamo pa tudi Newtonovo bazo, ki ni nič drugega kot posplošitev potenčne

$$P_n = \mathcal{L}(\{(x-x_0)(x-x_1)\dots(x-x_{i-1})\}_{i=0}^n).$$

Prvi element v tem zapisu bi potem moral biti $(x - x_{-1})$, če bi se striktno držali formule. Toda ker ta točka ne obstaja, rečemo kar da je ta element konstanta 1. Prednost polinomske interpolacije je, da lahko interpolacijski polinom zapišemo v zaključeni obliki. To lahko izkoristimo za reševanje drugih numeričnih problemov, kot so odvajanje, integriranje in reševanje diferencialnih enačb. Lagrangeeva oblika interpolacijskega polinoma je

$$p = \sum_{i=0}^{n} f(x_i)\ell_i,$$

za neke točke $a = x_0 < x_1 < \cdots < x_n = b$, Newtonovo obliko pa bomo spoznali nekoliko kasneje.

Dva primera bolj pomembnih baz za našo uporabo pa sta baza linearnih odrezanih potenc in baza z lokalnimi nosilci, ki pa ju bomo spoznali v naslednjem razdelku.

Pomemben del interpolacije je tudi pojem deljenih diferenc, ki jih bomo potrebovali pri izpeljavi predpisa za B-zlepke.

Definicija 3.1 *Naj bo $p \in P_k$ interpolacijski polinom stopnje $\leq k$, ki se s funkcijo f ujema v točkah $x_i, x_{i+1}, \dots, x_{i+k}$. Vodilni koeficient polinoma p je deljena diferenca funkcije f in jo označimo $[x_i, x_{i+1}, \dots, x_{i+k}]f$.*

Deljeno diferenco lahko zapišemo tudi v zaključeni oblki. Če so točke $x_i, x_{i+1},$

$\ldots, x_{i+k}$ med seboj različne, potem jo lahko zapišemo takole:

$$[x_i, x_{i+1}, \ldots, x_{i+k}]f = \sum_{j=i}^{i+k} f(x_j)[x_i, x_{i+1}, \ldots, x_{i+k}]\ell_j = \sum_{j=i}^{i+k} \frac{f(x_j)}{\prod_{\substack{r=i \\ r\neq j}}^{i+k}(x_j - x_r)} \tag{3.1}$$

Argumente v oglatih oklepajih lahko poljubno menjamo. Polinom je namreč enolično določen s temi točkami. Zato lahko brez problema kakšno točko zamenjamo, saj bo interpolacijski polinom tako ali tako moral iti skozi vse te točke.

Ni pa nujno, da so točke med seboj različne. Kakšne točke se lahko ponovijo večkrat, takrat govorimo o večkratnem ujemanju funkcij. Če se interpolacijski polinom z $f \in C^k([a,b])$ v točki $x_i = x_{i+j} \in [a,b], j = 1, 2, \ldots, k$ ujema $(k+1)$-kratno, potem je ta polinom kar Taylorjev s predpisom

$$p(x) = \sum_{j=0}^{k} \frac{f^{(j)}(x_i)}{j!}(x - x_i)^j,$$

deljena diferenca pa je

$$[\underbrace{x_i, x_i, \ldots, x_i}_{\text{j+1}}]f = \frac{1}{j!}f^{(j)}(x_i), j = 0, 1, \ldots, k.$$

V taki obliki se deljenih diferenc načeloma ne računa (razen za večkratno ujemanje, saj je dovolj enostavna). V ta namen se uporablja rekurzivna formula.

Izrek 3.2 *Naj bo $f \in C^k([a,b])$ in $x_j \in [a,b]$, $j = i, i+1, \ldots, i+k$. Tedaj je*

$$[x_i, \ldots, x_{i+k}]f = \frac{1}{k!}f^{(k)}(x_i) \tag{3.2}$$

če $x_i = x_{i+1} = \cdots = x_{i+k}$ in

$$[x_i, \ldots, x_{i+k}]f = \frac{[x_i, \ldots, x_{s-1}, x_{s+1}, \ldots, x_{i+k}]f - [x_i, \ldots, x_{r-1}, x_{r+1}, \ldots, x_{i+k}]f}{x_r - x_s}, \tag{3.3}$$

če $x_r \neq x_s$ in sta vsaj dve točki različni

Povedano z besedami:

- če sta dve ali več točk enakih, potem uporabimo formulo (3.2)
- če sta vsaj dve različni, uporabimo formulo (3.3): izberemo si dve točki izmed $x_i, \ldots, x_{i+k}$, ju vzamemo ven iz oklepajev (vsako samo enkrat), odštejemo dobljeni deljeni diferenci in delimo z razliko teh dveh točk

Na tem mestu lahko povemo še Newtonovo obliko polinoma, in sicer je

$$f(x) = \sum_{i=0}^{n} (x - x_0)(x - x_1) \ldots (x - x_{i-1})[x_0, x_1, \ldots, x_i]f.$$

Deljene diference se računa s tabelo, kar nam nekoliko pohitri izračun. Najlažje se to pokaže kar s primerom.

Primer: Naj bo $f(x) = \ln(x)$, interpolirali pa jo bomo s kubičnim polinomom na intervalu $[1, 2]$ z zaporedjem točk $x_1 = 1, x_2 = 1, x_3 = 2, x_4 = 2$. Deljene diference se računa po formulah zgoraj, vedno pa se začne z najmanjšo diferenco, torej z eno točko. Deljena diferenca $[x_i]f$ je enaka vrednosti funkcije v tisti točki $f(x_i)$. Ostale se nato računajo po formuli. Torej če sta točki enaki, pogledamo odvod izračunan v tisti točki. Če sta različni, vsako posebej vzamemo iz diference in delimo z njuno razliko.

	$[\cdot]f$	$[\cdot,\cdot]f$	$[\cdot,\cdot,\cdot]f$	$[\cdot,\cdot,\cdot,\cdot]f$
1	$\underline{0}$ ↘			
		$\underline{1}$ ↘		
1	0 ↗		$\underline{-0,306853}$ ↘	
		$0,693147$ ↗		$\underline{0,113706}$
2	$0,693147$		$-0,193147$ ↗	
		$0,5$		
2	$0,693147$			

Tabela 3.2: Tabela deljenih diferenc

Tabela 3.2 prikazuje izračunane deljene diference za ta primer.

V prvem stolpcu računamo samo deljene diference na eni stični točki, v drugem na dveh, v tretjem na treh in tako naprej. Tabela nam pomaga pri izbiri točk za izračun željene deljene diference. Če vzamemo na primer $[1,1,2]f$, bomo izbrali točke 1 in 2, saj imamo že izračunani $[1,1]f$ in $[1,2]f$. Enako bomo naredili pri izračunu $[1,1,2,2]f$, kjer bomo uporabili že izračunani $[1,1,2]f$ in $[1,2,2]f$. Pot računanja je nakazana s puščicami. Poleg tega, da uporabimo sistematičen način računanja, nam tudi ni potrebno izračunati vseh deljenih diferenc. Tako smo morali izračunati samo 3 na dveh točkah, vseh pa je 6. V tem primeru so slučajno samo tri različne, toda v splošnem temu ni tako.

Treba je še poudariti, da bi lahko tudi zamenjali vrstni red točk, ki smo jih dajali ven iz diference. Na primer, ko smo računali $[1,2,2]f$, bi lahko najprej dali ven 1 in šele nato 2 in odšteli, lahko pa bi najprej dali ven 2 in nato 1, rezultat pa se ne bi spremenil, saj se tudi vrstni red v imenovalcu spremeni in predznak ostane enak.

Iz tabele pa sedaj lahko enostavno preberemo koeficiente interpolacijskega polinoma. Bazne funkcije so v tem primeru

$$1, x-1, (x-1)^2, (x-1)^2(x-2),$$

če se držimo Newtonove oblike polinoma, koeficienti pa so zgornje vrednosti stolpcev. Iz tabele 3.2 so te vrednosti podčrtane. Polinom se torej glasi

$$p(x) = 1 * 0 + (x-1) - 0,306835(x-1)^2 + 0.113706(x-1)^2(x-2)$$

Za nas je glede deljenih diferenc pomembno še Leibnizevo pravilo, ki ga bomo uporabili v izpeljavi B-zlepkov.

Izrek 3.3 *Naj bo f produkt dveh funkcij, torej $f = g \cdot h$. Tedaj velja Leibnizevo pravilo*

$$[x_i, \ldots, x_{i+k}]f = \sum_{\ell=i}^{i+k} [x_i, \ldots, x_\ell]g[x_\ell, \ldots, x_{i+k}]h.$$

Dokaz bomo izpustili, najdemo ga pa lahko v [4](str. 87).

3.2 Odsekoma polinomske funkcije

Bolj pomemben prostor za naš problem je prostor odsekoma polinomskih funkcij. V tem prostoru se nahajajo tiste funkcije, ki so razdeljene na odseke in so na vseh odsekih polinomske. Ni nujno da so definirane na celi realni osi (ponavadi samo na nekem delu) in tudi ni nujno da so zvezne. Za boljšo aproksimacijo je pri takih funkcijah bolj smiselno večati število odsekov kot pa stopnjo polinoma. Ponavadi se vzame stopnja na primer 3 ali 4.

Definicija 3.4 *Naj bo* $[a, b] \in \mathbb{R}$ *dan interval z delitvijo*

$$a = x_0 < x_1 < \cdots < x_n = b.$$

Točkam $x_i, \quad i = 0, 1, \ldots, n$ *rečemo stične točke.*

V stičnih točkah se stikajo posamezni polinomski odseki. Velikost delitve pa je definirana kot največja velikost podintervala od ene stične točke do druge.

Sosednje odseke lahko povežemo z določenimi pogoji. Lahko zahtevamo, da je prvih $\nu_i - 1$ odvodov zveznih, kjer smo z ν_i označili število pogojev. Najmanj pogojev je 0, največ pa $k + 1$, kjer je k stopnja polinoma. Če je $\nu_i = 0$, to pomeni, da mora biti prvih $\nu_i - 1 = 0 - 1 = -1$ odvodov zveznih. Ker minus prvi odvod ne obstaja, pomeni da funkcija v točki x_i ni zvezna. Če je $\nu_i = 1$, potem to pomeni da mora biti ničti odvod zvezen, oziroma da je funkcija zvezna v x_i. Tako gre vse naprej do $\nu_i = k + 1$, kar pomeni da se sosednja odseka popolnoma ujemata, saj je stopnja polinoma k. V tem primeru lahko stično točko odstranimo.

Smiselno je vpeljati še pojem vozla. To je konstrukt, ki ga v našem algoritmu najbolj uporabljamo. Vozel je stična točka, šteta s kratnostjo, ki jo navežemo na število pogojev med sosednjima odsekoma. Vozel je r-kraten

takrat, ko ima funkcija v stični točki $k - r$ zveznih odvodov. Na primer, če vzamemo $\nu_i = 0$, pomeni da ima polinom v x_i -1 zveznih odvodov. Če obrnemo enačko $k-r = -1$ dobimo $r = k+1$, torej je vozel $k+1$-kraten. Če zahtevamo navadno zveznost ($\nu_i = 1$), je vozel zato k-kraten in tako naprej. Vozel je lahko tudi 0-kraten, in sicer v primeru ko je $\nu_i = k + 1$. Tedaj je $k - r = k + 1 - 1$ in $r = 0$. Vozel, tako kot stična točka, v tem primeru ne obstaja.

V [4] je narejen odličen primer, ki ga bomo navedli tudi tukaj.
Primer: Vzemimo funkcijo

$$f(x) = \begin{cases} 9 - x^3, & 0 \leq x < \frac{3}{2}, \\ x^3 + \frac{1}{2}, & \frac{3}{2} \leq x < 2, \\ -\frac{1}{2}x^3 + x^2 + \frac{17}{2}, & 2 \leq x < 3, \\ 2x^3 - 18x^2 + \frac{93}{2} - \frac{55}{2}, & 3 \leq x \leq 5.s \end{cases}$$

Funkcija je odsekoma polinomska, stopnje tri in s štirimi odseki. Iz definicije preberemo stične točke $x_0 = 0, x_1 = \frac{3}{2}, x_2 = 2, x_3 = 3, x_4 = 5$. Sedaj moramo dobiti še ν_1, ν_2 in ν_3, saj v točkah x_0 in x_5 ni sosednjih odsekov. Število pogojev bomo dobili tako, da pogledamo funkcijo v robnih točkah in enkrat vstavimo v levi odsek nato pa še v desnega. Ko bomo vstavili v levi odsek bomo označili z $f(\cdot - 0)$, za desni odsek pa $f(\cdot + 0)$. Poglejmo za prvo stično točko $x_1 = \frac{3}{2}$:

$$f(\frac{3}{2} - 0) \neq f(\frac{3}{2} + 0),$$

torej je $\nu_1 = 0$. Za stično točko $x_2 = 2$ velja:

$$f(2 - 0) = f(2 + 0), f'(2 - 0) \neq f'(2 + 0),$$

torej se ujemata v funkcijski vrednosti, ne pa v prvem odvodu in zato je $\nu_2 = 1$. V zadnji stični točki $x_3 = 3$ pa velja

$$f(3 - 0) = f(3 + 0), f'(3 - 0) = f'(3 + 0), f''(3 - 0) \neq f''(3 + 0),$$

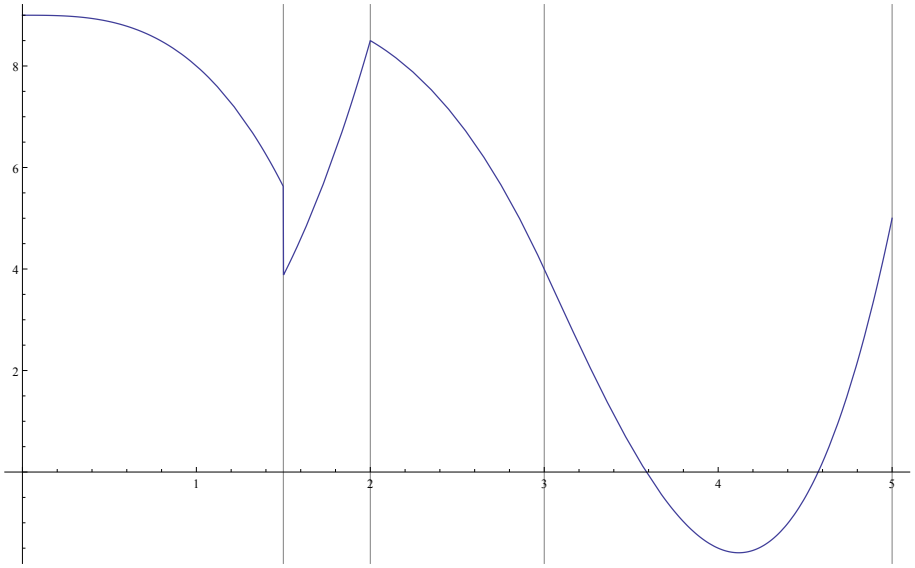
torej je šele pri drugem odvodu različna vrednost in zato je $\nu_3 = 2$.
Vemo, da je polinom stopnje 3 in zato lahko napišemo zaporedje vozlov:

$$0, \frac{3}{2}, \frac{3}{2}, \frac{3}{2}, \frac{3}{2}, 2, 2, 2, 3, 3, 5$$

V prvi in zadnji stični točki ne moremo govoriti o pogojih zveznosti, saj ležita na robu definicijskega območja. Za $x_1 = \frac{3}{2}$ lahko izračunamo večratnost kot

$$\begin{aligned} k - r &= -1 \\ 3 - r &= -1 \\ r &= 3 + 1 = 4, \end{aligned}$$

torej se ta vozel napiše štirikrat. Podobno lahko vidimo za ostala dva. Graf funkcije je prikazan na sliki 3.1.



Slika 3.1: Primer odsekoma polinomske funkcije

Odsekoma polinomskim funkcijam, za katere velja

$$\nu_i = k, \quad i = 1, 2, \ldots, n-1,$$

rečemo zlepki. Ker velja zgornji enačaj za vse stične točke, lahko izračunamo kratnost:

$$\begin{aligned} k - r &= \nu_i - 1 \\ k - r &= k - 1 \\ r &= 1 \end{aligned}$$

Kratnost vseh stičnih točk je tako 1 (krajišči imata že tako ali tako kratnost enako 1) in zato sta v tem primeru izraza stična točka in vozel ekvivalentna.

Poznamo dva naravna pristopa za izražavo odsekoma polinomskih funkcij. Prva je po intervalih - odsekoma, druga pa z uporabo baze prostora zlepkov $S_{k,\boldsymbol{x}}$, kjer k označuje stopnjo polinoma, $\boldsymbol{x}$ pa vektor vozlov. Za nas pride v poštev druga možnost, torej uporaba baze B-zlepkov.

3.3 B-zlepki

Za prostor $S_{1,\boldsymbol{x}}$ smo že omenili dve uporabni bazi. Bazo odrezanih potenc in bazo z lokalnimi nosilci.

Stične točke naj bodo definirane kot $x_0 < x_1 < \cdots < x_n$. Baza linearnih odrezanih potenc je tako

$$1, (x - x_i)_+, \quad i = 0, 1, \ldots, n-1.$$

Odrezana potenca stopnje n je definirana kot

$$z_+^n := \max(z, 0)^n.$$

Če vzamemo $n = 1$, je potem $z_+^1 = \begin{cases} 0, & z < 0, \\ z, & z > 0. \end{cases}$

Baza z lokalnimi nosilci je definirana kot

$$H_i(x) = \begin{cases} \frac{x - x_{i-1}}{x_i - x_{i-1}}, & x_{i-1} < x < x_i, \quad 0 < i, \\ 1, & x = x_i, \\ \frac{x_{i+1} - x}{x_{i+1} - x_i}, & x_i < x < x_{i+1}, \quad i < n, \\ 0, & \text{sicer}, \end{cases} \tag{3.4}$$

kjer i teče od 0 do n. Pri tej bazi je smiselno še definirati pojem lokalnega nosilca

$$supp(f) = \{x \in X \mid f(x) \neq 0\},$$

kjer je $X \subset \mathbb{R}$ in $f : X \to \mathbb{R}$. Pri bazi H_i so lokalni nosilci vsebovani v intervalu $[x_{i-1}, x_{i+1}]$. Nekaj podobnega želimo narediti za bazo prostora $S_{k,\boldsymbol{x}}$ in izkaže se, da lahko naredimo naravno posplošitev baze H_i na splošen k. H_i lahko zapišemo tudi kot

$$H_i(x) = (x_{i+1} - x_{i-1})[x_{i-1}, x_i, x_{i+1}](\cdot - x)_+ \tag{3.5}$$

Preverimo sedaj, če to res drži. Najprej izračunajmo deljeno diferenco po formuli (3.3):

$$[x_{i-1}, x_i, x_{i+1}](\cdot - x)_+ = \frac{[x_i, x_{i+1}](\cdot - x)_+ - [x_{i-1}, x_{i+1}](\cdot - x)_+}{x_i - x_{i-1}} =$$

$$= \frac{\frac{(x_i - x)_+ - (x_{i+1} - x)_+}{x_i - x_{i+1}} - \frac{(x_{i-1} - x)_+ - (x_i + 1 - x)_+}{x_{i-1} - x_{i+1}}}{x_i - x_{i-1}}$$

Sedaj ločimo 3 primere:

i. $x_{i-1} < x < x_i$

Upoštevamo definicijo odrezane potence za $n = 1$ in dobimo

$$\frac{\frac{x_i - x - x_{i+1} + x}{x_i - x_{i+1}} - \frac{0 - x_{i+1} + x}{x_{i-1} - x_{i+1}}}{x_i - x_{i-1}} = \frac{1 - \frac{x - x_{i+1}}{x_{i-1} - x_{i+1}}}{x_i - x_{i-1}} =$$

$$\frac{x_{i-1} - x_{i+1} - x + x_{i+1}}{(x_i - x_{i-1})(x_{i-1} - x_{i+1})} =$$

$$= \frac{-x + x_{i-1}}{(x_i - x_{i-1})(x_{i-1} - x_{i+1})}$$

Če sedaj še pomnožimo z $(x_{i+1} - x_{i-1})$, dobimo to kar smo hoteli, torej

$$\frac{x - x_{i-1}}{x_i - x_{i-1}}$$

ii. $x = x_i$
Kot prej upoštevamo definicijo odrezane potence:

$$\frac{\frac{0 - x_{i+1} + x}{x - x_{i+1}} - \frac{0 + x - x_{i+1}}{x_{i-1} - x_{i+1}}}{x - x_{i-1}} = \frac{x_{i-1} - x_{i+1} - x + x_{i+1}}{(x - x_{i-1})(x_{i-1} - x_{i+1})} =$$

$$= \frac{1}{x_{i+1} - x_{i-1}}$$

Očitno smo dobili pravilen izraz, saj dobimo 1, če pomnožimo z istim oklepajem kot prej.

iii. $x_i < x < x_{x+1}$
V zadnjem primeru ponovimo to, kar smo naredili prej:

$$\frac{\frac{0 + x - x_{i+1}}{x_i - x_{i+1}} - \frac{0 + x - x_{i+1}}{x_{i-1} - x_{i+1}}}{x_i - x_{i-1}} =$$

$$= \frac{(x - x_{i+1})(x_{i-1} - x_{i+1}) - (x - x_{i+1})(x_i - x_{i+1})}{(x_i - x_{i-1})(x_i - x_{i+1})(x_{i-1} - x_{i+1})} =$$

$$= \frac{(x - x_{i+1})(x_{i-1} - x_{i+1} - x_i + x_{i+1})}{(x_i - x_{i-1})(x_i - x_{i+1})(x_{i-1} - x_{i+1})} =$$

$$= \frac{x - x_{i+1}}{(x_i - x_{i+1})(x_{i+1} - x_{i-1})}$$

Spet pomnožimo z istim oklepajem kot v prvih dveh primerih. Ta ulomek lahko potem zgoraj in spodaj delimo s tem oklepajem in dobimo pravilen rezultat. Če želimo, lahko tudi obrnemo vse predznake ulomka, da dobimo identičen ulomek kot v (3.4).

Lokalne nosilce H_i smo indeksirali s srednjo stično točko. Če pa želimo posplošiti definicijo za splošen k, pa to ni najbolje, saj ni nujno da ta točka obstaja. Poleg tega nastopijo problemi, če izberemo $i = 0$. Takrat moramo definirati še x_{-1}, ponavadi pa se to naredi kar tako, da rečemo da je razmik $x_0 - x_{-1} = 0$. V izrazu (3.5) to pomeni, da smo eno točko šteli dvojno, kar pa potem podere izpeljavo zgoraj. Zato bomo raje indeksirali s prvo točko. Ni pa nujno, da se v splošnem to zgodi samo na robu. Posplošitev bomo zato definirali z zaporedjem vozlov in ne stičnih točk. Tako dobimo definicijo baznega oziroma B-zlepka:

$$B_{i,k}(x) = (t_{i+k+1} - t_i)[t_i, \ldots, t_{i+k+1}](\cdot - x)_+^k \tag{3.6}$$

Če so vsi vozli med seboj različni, lahko uporabimo (3.1) in zapišemo zaključeno obliko B-zlepka

$$B_{i,k}(x) = (t_{i+k+1} - t_i) \sum_{\ell=i}^{i+k+1} \frac{(t_\ell - x)_+^k}{\prod_{\substack{j=i \\ j\neq\ell}}^{i+k+1}(t_\ell - t_j)}. \tag{3.7}$$

Tako kot pri bazi z lokalnimi nosilci tudi tukaj velja, da so nosilci B-zlepka vsebovani v $[t_i, t_{i+k+1}]$, torej velja

$$supp B_{i,k} \subset [t_i, t_{i+k+1}]. \tag{3.8}$$

Povedano nekoliko drugače, to pomeni, da je funkcija neničelna samo na tem intervalu, povsod drugje pa je identično enaka 0.

Zaključene oblike B-zlepka (3.7) v veliki večini primerov ne uporabljamo za izračun vrednosti. Zato bomo sedaj izpeljali boljši, numerično absolutno stabilen, način izračuna.

Za odrezane potence velja zveza

$$(\cdot - x)_+^k = (\cdot - x)_+(\cdot - x)_+^{k-1} = (\cdot - x)(\cdot - x)_+^{k-1}$$

za $k \geq 1$. Hitro lahko preverimo, da to velja glede na definicijo odrezane

potence. Če je $(\cdot - x) > 0$ se brez problemov lahko razdeli na več oklepajev, saj bo oklepaj ostal tak kot je. Če pa je $(\cdot - x) < 0$, bo pa prvi oklepaj neka negativna vrednost, drugi oklepaj pa bo vrnil 0, torej bo cel izraz enak 0. Vstavimo sedaj to v (3.6) in dobimo

$$B_{i,k}(x) = (t_{i+k+1} - t_i)[t_i, \ldots, t_{i+k+1}]((\cdot - x)(\cdot - x)_+^{k-1})$$

Z upoštevanjem izreka 3.3 pa dobimo

$$\begin{gathered} B_{i,k}(x) = (t_{i+k+1} - t_i)(([t_i](\cdot - x))([t_i, \ldots, t_{i+k+1}](\cdot - x)_+^{k-1}) + \\ +([t_i, t_{i+1}](\cdot - x))([t_i + 1, \ldots, t_{i+k+1}](\cdot - x)_+^{k-1}) + 0) = \\ = (t_{i+k+1} - t_i)((t_i - x)([t_i, \ldots, t_{i+k+1}](\cdot - x)_+^{k-1}) + \\ +[t_{i+1}, \ldots, t_{i+k+1}](\cdot - x)_+^{k-1}) \end{gathered}$$

Iz vsote dobimo samo 2 člena, saj je $(\cdot - x)$ linearna funkcija, za te pa so neničelne samo deljene diference na eni ali dveh točkah. Izračunajmo sedaj deljeno diferenco po formuli (3.3)

$$\begin{gathered} [t_i, \ldots, t_{i+k+1}](\cdot - x)_+^{k-1} = \\ = \frac{1}{t_i - t_{i+k+1}}([t_i, \ldots, t_{i+k}](\cdot - x)_+^{k-1}) - [t_{i+1}, \ldots, t_{i+k+1}](\cdot - x)_+^{k-1} = \\ = \frac{1}{t_i - t_{i+k+1}}\Big(\frac{B_{i,k-1}(x)}{t_{i+k} - t_i} - \frac{B_{i+1,k-1}(x)}{t_{i+k+1} - t_{i+1}}\Big) \end{gathered}$$

Drugi enačaj velja, če upoštevamo enačbo (3.6) za dane argumente. Če sedaj to deljeno diferenco vstavimo v zgornjo enačbo B-zlepka, pa dobimo

$$B_{i,k}(x) = (t_{i+k+1} - t_i)\left(\frac{t_i - x}{t_i - t_{i+k+1}}\Big(\frac{B_{i,k-1}(x)}{t_{i+k} - t_i} - \frac{B_{i+1,k-1}(x)}{t_{i+k+1} - t_{i+1}}\Big) + \frac{B_{i+1,k-1}(x)}{t_{i+k+1} - t_{i+1}}\right)$$

Sedaj lahko še strnemo izraz v obliko brez oklepajev in dobimo rekurzivno formulo v končni obliki:

$$B_{i,k}(x) = \frac{x - t_i}{t_{i+k} - t_i} B_{i,k-1}(x) + \frac{t_{i+k+1} - x}{t_{i+k+1} - t_{i+1}} B_{i+1,k-1}(x), \tag{3.9}$$

robni pogoj pa je

$$B_{i,0}(x) = \begin{cases} 1, & t_i \leq x < t_{i+1} \ \ ali \ \ x = t_{i+1} = t_n, \\ 0, & \text{sicer.} \end{cases}$$

Zaradi (3.8) sta obe uteži (ulomka) v (3.9) nenegativni in v primeru, ko je x iz tega intervala, celo pozitivni. To nas pripelje do še ene lastnosti, in sicer B-zlepek je strogo pozitiven na odprtem intervalu (t_i, t_{i+k+1}). Zaradi te lastnosti in rekurzivne formule, rečemo, da so B-zlepki neobčutljiva baza $S_{k,\boldsymbol{x}}$.

Zaporedje vozlov, ki definirajo B-zlepek pa dobimo tako, da stičnim točkam dodamo še k točk na začetku in na koncu. Vseh vozlov je tako $n + 2k + 1$:

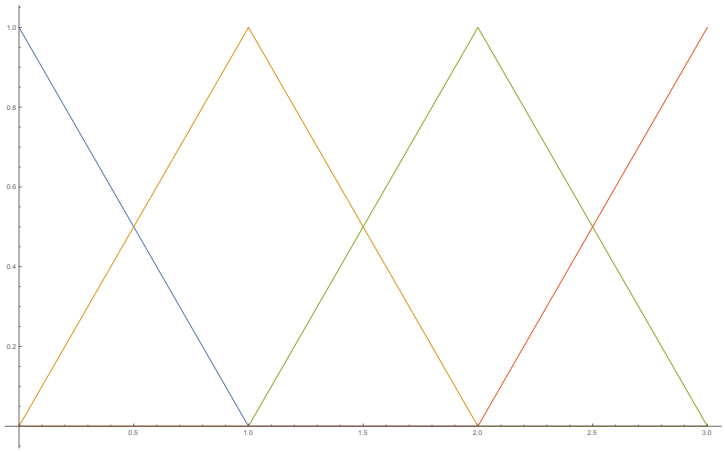
$$\boldsymbol{t} = (\underbrace{x_0, \ldots, x_0}_{\text{k+1}}, x_1, x_2, \ldots, x_{n-1}, \underbrace{x_n, \ldots, x_n}_{\text{k+1}})$$

Vozle štejemo od 1 naprej in zato raje prepišemo zgornji vektor:

$$\boldsymbol{t} = (t_1, t_2, \ldots, t_{n+2k+1})$$

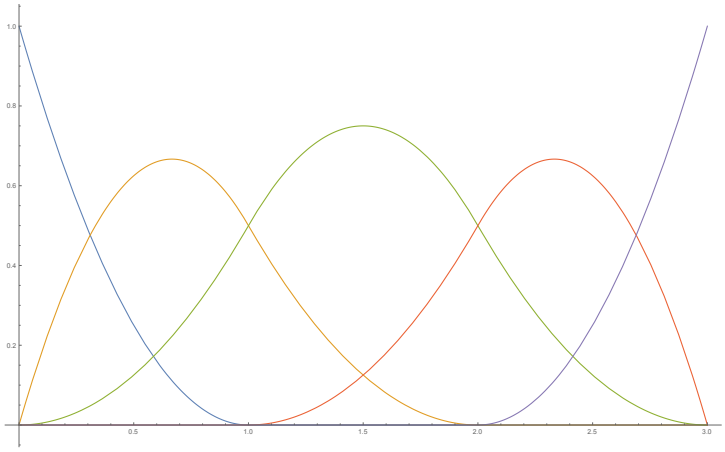
Vseh različnih B-zlepkov za neko stopnjo k je zato $n + k = \dim S_{k,\boldsymbol{x}}$. Za primer si poglejmo nekaj baz za različne stopnje in stične točke.

Primer: Za prvi primer si poglejmo stopnjo $k = 1$ za stične točke $\boldsymbol{x} = (0, 1, 2, 3)$. Vseh B-zlepkov v tem prostoru je $n + k = 4$. To bazo poznamo tudi kot bazo z lokalnimi nosilci H_i.



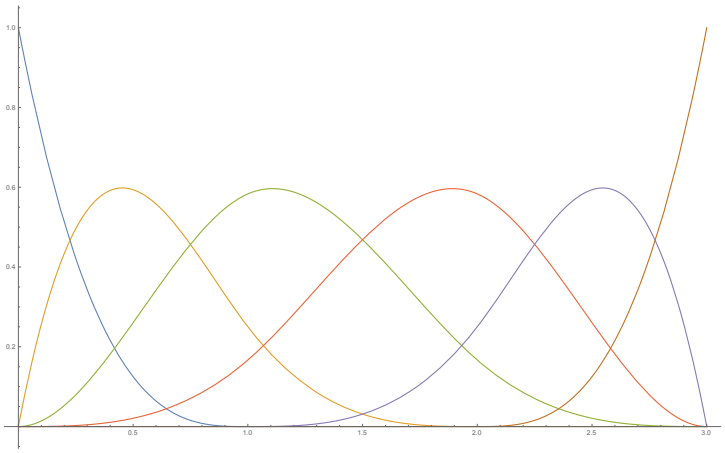
Slika 3.2: Vsi B-zlepki stopnje 1 na štirih točkah

Primer: Drug primer je podoben prvemu, samo da vzamemo stopnjo $k = 2$ in zato je B-zlepkov seveda več, in sicer $n + k = 5$.



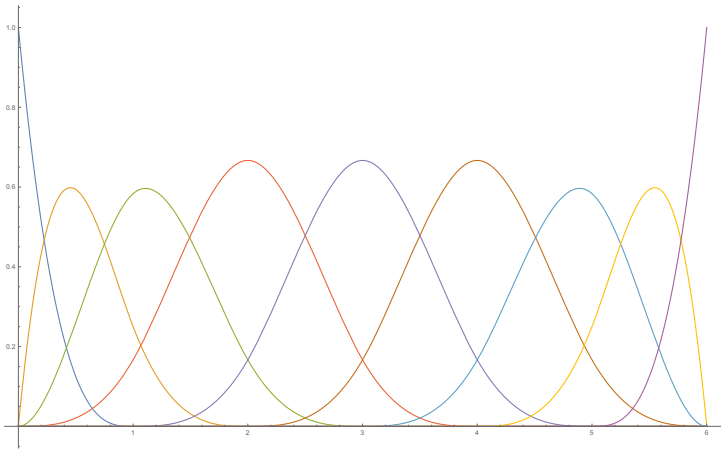
Slika 3.3: Vsi B-zlepki stopnje 2 na štirih točkah

Primer: Kot prejšnji primer je ta narejen na štirih stičnih točkah, spet pa smo povečali stopnjo na $k = 3$. B-zlepkov je zato $n + k = 6$.



Slika 3.4: Vsi B-zlepki stopnje 3 na štirih točkah

Primer: Sedaj pa lahko še povečamo število stičnih točk na 7, in sicer $\boldsymbol{x} = (0, 1, 2, 3, 4, 5, 6)$. Stopnja naj ostane ista, število B-zlepkov pa se spet poveča na $n + k = 9$.



Slika 3.5: Vsi B-zlepki stopnje 3 na sedmih točkah

Pri vseh primerih moramo upoštevati, da smo stične točke vedno indeksirali od 0 naprej in zato jih je vedno $n + 1$.

Poznamo še en pomemben izrek, ki opredeli korektnost interpolacijskih nalog v prostoru odsekoma polinomskih funkcij.

Izrek 3.5 *(Schoenberg-Whitney) Zlepek $s \in S_{k,\boldsymbol{x}}$ je z interpolacijskimi pogoji*

$$s(\tau_i) = f_i, \quad i = 1, 2, \ldots, n+k,$$

kjer so interpolacijske točke τ_i urejene $\tau_1 < \tau_2 < \cdots < \tau_{n+k}$, enoličnо določen natanko tedaj, ko je $\tau_i \in (t_i, t_{i+k+1})$ za vse i.

Za interpolacijo neke funkcije f, v našem primeru polinoma, z B-zlepki moramo poiskati koeficiente $\alpha_i \in \mathbb{R}$, za katere bo veljalo

$$f(x) = \sum_{i=1}^{n+k} \alpha_i B_{i,k}(x).$$

Iščemo torej neko linearno kombinacijo baznih zlepkov.

Poglavje 4

Algoritem

4.1 Teorija za algoritmom

Algoritem, ki ga bomo razvili, uporablja še pojem kontrolnega poligona. Tudi to je odsekoma polinomska funkcija, stopnja posameznih odsekov pa je striktno enaka 1. Lahko bi rekli, da je odsekoma linearna funkcija ali lomljenka. Označimo jo z

$$\Gamma = \Gamma(f).$$

Temelji na enakih vozlih, ki definirajo B-zlepek, točke te funkcije pa so

$$(\bar{t}_i, \alpha_i)_{i=1}^{n+k},$$

posamezne $\bar{t}_i$ imenujemo i-to povprečje vozlov, definirani pa so takole:

$$\bar{t}_i = \frac{1}{k} \sum_{j=i+1}^{i+k} t_j.$$

Tako dobimo zaporedje $n + k$ točk, ki jih med sabo povežemo in dobimo kontrolni poligon Γ. Ta že sam po sebi nekoliko aproksimira funkcijo, ima pa nekaj uporabnih lastnosti, ki jih lahko s pridom izkoriščamo. Ena izmed teh lastnosti pravi, da ima lahko zlepek največ toliko ničel, kot jih ima njegov kontrolni poligon.

Pomemben je tudi izračun ničle kontrolnega polinoma. To je lažje, saj gre za linearno funkcijo. Označimo jo

$$y = ax + b,$$

poiskati pa moramo koeficient a in začetno vrednost b. Ker v našem algoritmu za privzeto vrednost uporabljamo stopnjo polinoma $k = 3$ in zaradi preglednosti, bomo izpeljavo pokazali na tej stopnji.

Naj se ničla kontrolnega polinoma nahaja med točkama $(\bar{t}_i, \alpha_i)$ in $(\bar{t}_{i+1}, \alpha_{i+1})$. Koeficient a lahko izračunamo po znani formuli

$$a = \frac{\alpha_{i+1} - \alpha_i}{\bar{t}_{i+1} - \bar{t}_i}.$$

Vstavimo to v enačbo in upoštevajmo, da je ničla funkcije točka $(x, 0)$

$$0 = \frac{\alpha_{i+1} - \alpha_i}{\frac{t_{i+2}+t_{i+3}+t_{i+4}}{3} - \frac{t_{i+1}+t_{i+2}+t_{i+3}}{3}} x + b = \frac{3(\alpha_{i+1} - \alpha_i)}{t_{i+4} - t_{i+1}} x + b \tag{4.1}$$

$$x = -b \frac{t_{i+4} - t_{i+1}}{3(\alpha_{i+1} - \alpha_i)} \tag{4.2}$$

Kar nam še manjka je izračun začetne vrednosti b. Za to moramo vstaviti neko točko v enačbo (4.1), le da moramo vzeti za splošno točko, torej bo leva stran namesto 0 kar y. Vstavimo na primer točko $(\bar{t}_{i+1}, \alpha_{i+1})$. Računali bomo torej ničlo x_{i+1}, ki se nahaja med točkama, ki smo jih omenili zgoraj.

$$b = \alpha_{i+1} - \frac{3(\alpha_{i+1} - \alpha_i)}{t_{i+4} - t_{i+1}} \frac{t_{i+2} + t_{i+3} + t_{i+4}}{3} =$$

$$= \frac{\alpha_{i+1}(t_{i+4} - t_{i+1}) + (t_{i+2} + t_{i+3} + t_{i+4})(\alpha_i - \alpha_{i+1})}{t_{i+4} - t_{i+1}}$$

To sedaj vstavimo v enačbo (4.2) in dobimo končen izračun ničle kontrolnega polinoma:

$$x_{i+1} = -\frac{t_{i+4} - t_{i+1}}{3(\alpha_{i+1} - \alpha_i)} \frac{\alpha_{i+1}(t_{i+4} - t_{i+1}) + (t_{i+2} + t_{i+3} + t_{i+4})(\alpha_i - \alpha_{i+1})}{t_{i+4} - t_{i+1}} =$$

$$= -\frac{\alpha_{i+1}(t_{i+4} - t_{i+1})}{3(\alpha_{i+1} - \alpha_i)} + \frac{t_{i+2} + t_{i+3} + t_{i+4}}{3} = \bar{t}_{i+1} - \frac{\alpha_{i+1}(t_{i+4} - t_{i+1})}{3(\alpha_{i+1} - \alpha_i)}$$

Če upoštevamo splošno stopnjo polinoma, dobimo enačbo

$$x_{i+1} = \bar{t}_{i+1} - \frac{\alpha_{i+1}(t_{i+k+1} - t_{i+1})}{k(\alpha_{i+1} - \alpha_i)}, \tag{4.3}$$

lahko pa tudi zamaknemo vse indekse za -1, da dobimo obliko enačbe, ki smo jo uporabili v algoritmu.

Izredno pomemben del algoritma je tudi Boehm-ov algoritem, opisan v [1]. Ko najdemo ničlo kontrolnega polinoma, bomo prav to ničlo vstavili v zaporedje vozlov. Krivulja mora ostati taka kot je s še enim dodatnim vozlom in prav to nam ta algoritem omogoča. Po dodajanju vozla $\tilde{t}$ nam še spremeni vse koeficiente v linearni kombinaciji B-zlepkov. Napisano bolj formalno, če vzamemo trenutno stanje zlepka kot

$$\sum_{i=1}^{n+k} \alpha_i B_{i,k}(x)$$

z vozli

$$\boldsymbol{t} = (t_1, t_2, \ldots, t_{n+2k+1}),$$

bo novo stanje po vstavljanju vozla

$$\sum_{i=1}^{n+k+1} \tilde{\alpha}_i \tilde{B}_{i,k}(x)$$

z vozli

$$\boldsymbol{t} = (t_1, t_2, \ldots, t_l, \tilde{t}, t_{l+1}, \ldots, t_{n+2k+1}),$$

spremeni pa se tudi zaporedje koeficientov. Formula za nove koeficiente je

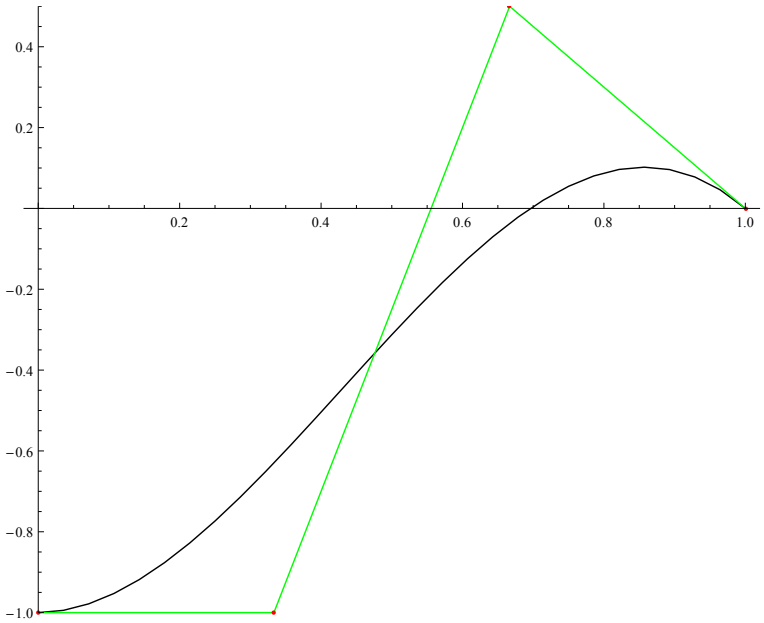
$$\tilde{\alpha}_i = (1 - \beta_i)\alpha_{i-1} + \beta_i \alpha_i,$$

kjer so

$$\beta_i = \begin{cases} 1, & i \leq l-k \\ 0, & i \geq l+1, \\ \frac{\tilde{t}-t_i}{t_{i+k}-t_i} & l-k+1 \leq i \leq l. \end{cases} \tag{4.4}$$

Povedano drugače, to pomeni, da bomo vse koeficiente, ki so na mestu manjšem ali enakem kot $l-k$, pustili pri miru, vse koeficiente, ki so večji od indeksa l, pa bomo zamaknili za eno mesto naprej. Ostanejo nam samo še tisti vozli, ki se nahajajo med mestoma $l-k+1$ in l, skupaj jih je k. Najlažje je ta algoritem prikazati na primeru.

Primer: Vzemimo zaporedje vozlov $\boldsymbol{t} = (0,0,0,0,1,1,1,1)$, zaporedje koeficientov $\boldsymbol{c} = (-1,-1,\frac{1}{2},0)$ in stopnjo $k = 3$. Začetna slika tega zlepka je prikazana na sliki 4.1. Vstavimo sedaj nov vozel $\tilde{t} = \frac{1}{2}$. Poiskati moramo



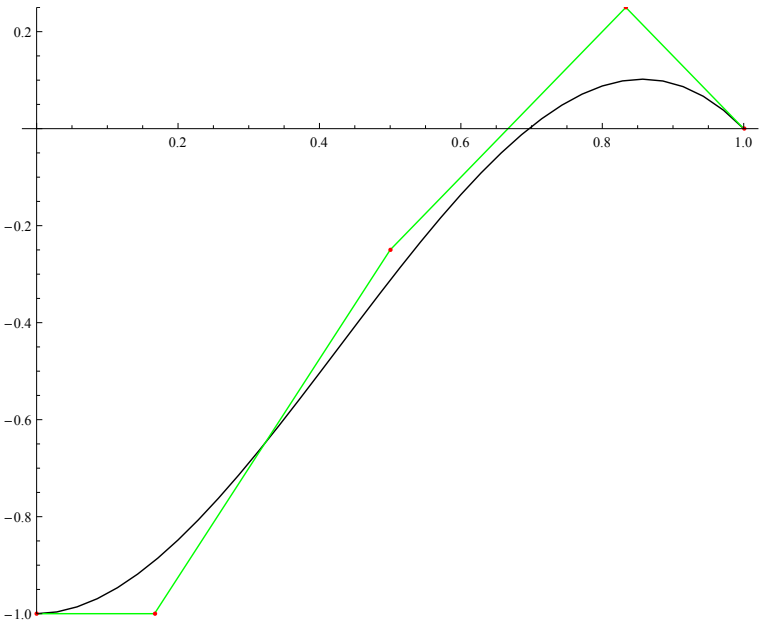
Slika 4.1: Slika B-zlepka pred vstavljanjem vozla

torej tak l, da bo $t_l < \tilde{t}$ in $t_{l+1} > \tilde{t}$. Ugotovimo, da se to zgodi pri $l = 4$, saj je $t_4 = 0$ in $t_5 = 1$. Novo zaporedje vozlov bo vsebovalo torej 9 elementov, koeficientov pa bo 5. Začnimo z računanjem novih koeficientov od zadaj, to pomeni od $n+k+1 = 5$, če upoštevamo, da imamo danih $n+1$ stičnih točk.

Najprej računajmo za $i = 5$, torej bomo morali izračunati β_5. Ker velja $5 = i \geq l + 1 = 5$, pademo v drugi primer formule (4.4) in bo zato $\beta_5 = 0$ in velja $\tilde{c}_5 = c_4 = 0$.
Naj bo sedaj $i = 4$. Sedaj pademo v tretji primer formule (4.4), saj velja $l - k + 1 = 2 \leq 4 \leq 4 = l$, torej $\beta_4 = \frac{\frac{1}{2}-0}{1-0} = \frac{1}{2}$. Izračunamo lahko $\tilde{c}_4 = \frac{1}{2}c_3 + \frac{1}{2}c_4 = \frac{1}{4}$.
Postopek ponovimo še za $i = 3$ in $i = 2$, dobimo pa rezultata $\tilde{c}_3 = -\frac{1}{4}$ in $\tilde{c}_2 = -1$. Za $i = 1$ pa velja $1 = i \leq l - k = 1$ in $\beta_1 = 1$, zato pride rezultat $\tilde{c}_1 = c_1 = -1$. Dobili smo nov vektor koeficientov

$$\tilde{\boldsymbol{c}} = (-1, -1, -\frac{1}{4}, \frac{1}{4}, 0).$$

Opazimo lahko, da se je v resnici spremenil samo en koeficient, en pa se je dodal. To je naključje, v splošnem velja, da se en doda, ostalih $k - 1$ pa se spremeni. Sedaj lahko narišemo isto krivuljo z dodatnim vozlom. Narisana je na sliki 4.2.



Slika 4.2: Slika B-zlepka po vstavljanju vozla $\frac{1}{2}$

V naši implementaciji je ta algoritem narejen v dveh korakih, in sicer najprej poskrbimo za prvi primer, za drugi primer dejansko ni potrebno nič narediti, šele nato pa računamo konstanto β.

Preden gremo na dejansko implementacijo moramo omeniti še kriterij za konvergenco. Ničle poligona se bodo nabirale ena za drugo, mi pa moramo preveriti če so približki dovolj skupaj da vrnemo vrednost kot ničlo polinoma. Kriterij za konvergenco je

$$\frac{\max_{j,k} |x_j - x_k|}{\max(|t_i|, |t_{i+d}|)} < \epsilon, \tag{4.5}$$

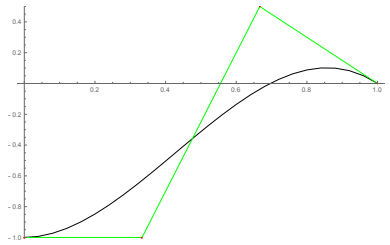
kjer moramo nujno upoštevati, da je maksimum v števcu vzet nad zadnjimi d približki. Iz tega lahko sklepamo, da moramo v seznam vozlov najprej vstaviti d vozlov, kjer smo z d označili stopnjo interpolacijskega polinoma, da lahko dobimo nek približek.

Ta metoda iskanja ničel je numerično stabilna, absolutno konvergentna, hitrost konvergence pa je kvadratična, če ima polinom samo enostavne ničle in linearna če so ničle višjih redov.

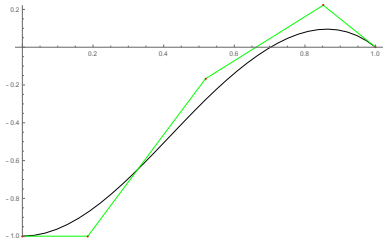
Metoda konvergira vedno, ko ima zlepek vsaj eno ničlo. Za prvi približek je smiselno vzeti ničlo kontrolnega poligona, saj, kot že rečeno, kontrolni poligon dobro aproksimira zlepek. Ko bomo vstavili en vozel, bo aproksimacija boljša, torej se bo poligon bolje ujemal z zlepkom. Ta postopek ponavljamo in izkaže se, da so ničle poligona vedno bolj podobne ničli zlepka. Zato vedno dobimo konvergenco, če ima zlepek vsaj eno ničlo. Več o tem si bralec lahko prebere v [5].

Zaključimo teorijo s primerom delovanja algoritma.

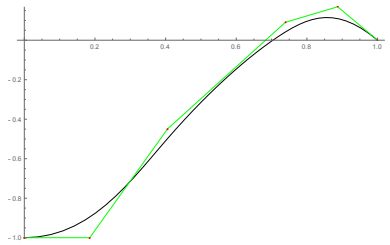
Primer: Vzemimo tako kot pri prejšnjem primeru zaporedje vozlov $\boldsymbol{t} = (0, 0, 0, 0, 1, 1, 1, 1)$, zaporedje koeficientov $\boldsymbol{c} = (-1, -1, \frac{1}{2}, 0)$ in stopnjo $k = 3$. Začetna slika tega B-zlepka je dana na 4.3.



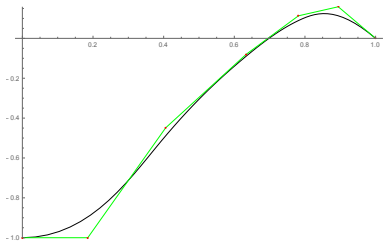
Slika 4.3: Začetna slika B-zlepka



Slika 4.4: Slika B-zlepka po prvi iteraciji



Slika 4.5: Slika B-zlepka po drugi iteraciji



Slika 4.6: Slika B-zlepka po tretji iteraciji

Že po prvi iteraciji se opazi izboljšano prileganje poligona polinomu. V drugih dveh se še bolj približa, na koncu pa lahko vidimo, da je po samo treh iteracijah ničla poligona že zelo blizu ničli polinoma.

4.2 Programska implementacija

Algoritem 4.1: Algoritem za iskanje vseh ničel polinoma

```
function [ zero ] = findZeros( f, a, b, d, max_it, eps )
%FINDZERO This function returns all zeros of a specified funciton.
%
% Input arguments are two kinds:
%   1. using a function
%     - an anonymous function e.g. '@(x) x.^4 + 6*x.^2 + 3*x'
%     - left edge of interval where zero should be searched
%     - right edge of the same interval
%   2. using knots and coefficients
%     - knot vector
%     - coefficient vector
%     - third argument must be [], if you wish to use optional parameters
% Both kinds support 3 optional arguments:
%     - (optional) degree of interpolation polynomials (default 4)
%     - (optional) maximum number of iterations (default 100)
%     - (optional) error tolerance (default 10e-15)
% Output is a vector of all zeros if they exist.

    if nargin < 2
        error('Not enough input arguments!');
    elseif nargin == 2
        b = [];
        d = 4;
        max_it = 100;
        eps = 1e-15;
    elseif nargin == 3
        d = 4;
        max_it = 100;
        eps = 1e-15;
    elseif nargin == 4
        max_it = 100;
        eps = 1e-15;
    elseif nargin == 5
        eps = 1e-15;
    end;


    if(isempty(b))
        t = f;
        c = a;
        d = d - 1;
    else
        if(b <= a)
            error('Left edge of interval must be a smaller integer than right edge!');
        end;

        interval = a:0.1:b;
        [~,sizeInt] = size(interval);
        spline = spap2(floor(sizeInt/2), d, interval, arrayfun(f,interval));

        t = spline.knots;
        c = spline.coefs;
        d = spline.order - 1;
    end;
    zero = [];
    if(abs(c(1)) <= eps)
        zero = [zero t(1)];
        return;
    end;
```

```
    listZeros = [];
    k = 2;
    ck = 0;
    for it = 1 : max_it
        [~,n] = size(c);
        while (k <= n) && (c(k-1) * c(k) >= 0)
            if(c(k-1) * c(k) == 0)
                ck = 1;
                break;
            else
                k = k + 1;
            end;
        end
        if k > n
            break;
        end;
        if(ck == 1)
            x = sum(t(k+1:k+d))/d;
        else
            % naredimo premico cez dve tocki in poiscemo niclo (ta enacba je izpeljava)
            x = sum(t(k+1:k+d))/d - (c(k) * (t(k+d) - t(k))) / ((c(k) - c(k-1)) * d);
        end;
        listZeros = [listZeros x];
                getVertices(t,c,d+1);
        if(converged(t, d, k, listZeros, eps) || ck == 1)
            ck = 0;
            zero = [zero x];
            if(k == n)
                break;
            end;
            it = 1;
            listZeros = [];
            k = k + 1;
        else
            [t,c] = refineSpline(t,c,k,n,d,x);
        end;
    end
    if(isempty(zero))
        zero = 'NO_ZERO';
    end;
end
```

Vsa izvorna koda je napisana v jeziku MATLAB s pomočjo psevdokode iz članka [5]. Požene se ga lahko na dva načina:

i. Podamo mu anonimno funkcijo f in interval $[a, b]$, na katerem bomo iskali ničle. Anonimna funkcija je oblike na primer $f = @(x)x^3 + x^2 + 2 * x - 3$. Ker sama interpolacija ni del tega diplomskega dela, smo to nalogo prepustili MATLAB-u.

ii. Podamo mu seznam vozlov in koeficientov α_i v zapisu funkcije v obliki B-zlepka.

Oba načina podpirata tudi opcijske parametre, in sicer stopnja polinoma,

število maksimalnih iteracij in ϵ iz (4.5). Pri drugem načinu zagona, pa moramo podati tretji parameter kot [ ], če želimo nastaviti opcijske parametre.

V programu je stopnja zlepka označena s črko d, čeprav smo v teoriji za to uporabljali črko k. Ta črka ima v našem programu nek drug pomen.

Prvi del programa poskrbi za inicializacijo opcijskih parametrov, če ti niso bili podani. Takoj za tem sledi drugi del, v katerem dobimo vektor vozlov in koeficientov. To naredimo na dva načina. Če smo pognali program na način *ii.*, potem preberemo ta dva vektorja iz parametrov. V nasprotnem primeru, ju moramo poiskati sami. V ta namen uporabimo MATLAB funkcijo „*spap*2“. Ta funkcija sprejme štiri argumente. Prvi je število željenih odsekov funkcije, drugi je stopnja odsekov, tretji je vektor vrednosti x, zadnji pa vektor vrednosti $f(x)$. Ta funkcija ne sprejme dejanskega predpisa polinoma, zato bo izvedel interpolacijo na točkah $(x, f(x))$. Že samo tukaj se lahko pojavijo numerične napake, saj bomo računali vrednosti podane anonimne funkcije, jih podali funkciji „*spap*2“, ta pa bo izvedel interpolacijo. Zato ne zagotavljamo popolne natančnosti za visoke stopnje polinomov. Spet poudarjamo, da sama interpolacija oziroma iskanje koeficientov α_i za linearno kombinacijo B-zlepkov ni del diplomskega dela.

V tretjem delu poteka glavnina našega programa. V vsaki iteraciji moramo najprej poiskati pravilen k, ki nam bo povedal med katerima vozloma se nahaja ničla. Lahko se zgodi, da tak k ne obstaja in je zato večji od števila koeficientov. V tem primeru zaključimo z izvajanjem programa. Sledi izračun števila x po enačbi (4.3), oziroma če smo ugotovili da je kakšen vozel sama ničla, izračunamo le i-to povprečje vozla. Nato dodamo x v seznam vseh približkov in preverimo konvergenco. To naredimo s pomočjo funkcije „*converged*“.

Algoritem 4.2: Funkcija za preverjanje konvergence približkov

```
function [ converged ] = converged( t, d, k, listZeros, eps )
%CONVERGED returns TRUE, if last element of listZeros is deemed to be the
%zero.
    [~,n] = size(listZeros);
    if n <= d + 1
        converged = false;
        return;
    elseif listZeros(n) == t(d+k)
        converged = true;
        return;
    end;

    maxKnot = max([abs(t(k)), abs(t(k+d))]);
    conv = [];
    for i = n-d-1:n-1
        for j = i+1:n
            absX = abs(listZeros(i) - listZeros(j));
            conv = [conv absX];
        end;
    end;

    maxAbsX = max(conv);

    if maxAbsX / maxKnot < eps
        converged = true;
    else
        converged = false;
    end;
end
```

Najprej je na vrsti preverjanje, da je bilo vstavljenih vsaj d vozlov. Če to ni res, funkcija takoj vrne „$false$“ in nadaljujemo s programom. Drugo preverjanje pa je prisotno zaradi povezanosti zlepka. Zlepek je lahko namreč tudi nepovezan. To se sicer zgodi zelo redko, vendar dodatno preverjanje ni odveč. V primeru da pridemo mimo obeh preverjanj, se ravnamo po (4.5). Izračunamo ulomek, preverimo če je manjši kot ϵ in vrnemo ustrezno vrednost.

V primeru da so približki konvergirali, torej je ta funkcija vrnila „$true$“, dodamo x v seznam vseh ničel, ponastavimo števec iteracij, počistimo seznam približkov in pomaknemo k za eno naprej (če ta ni enak številu koeficientov), da bomo iskali naslednje ničle. V nasprotnem primeru samo dodamo x v seznam vozlov in popravimo zlepek po Boehmovem algoritmu.

Algoritem 4.3: Boehmov algoritem

```
function [ t, c ] = refineSpline( knots, coefs, k, n, d, x )
%REFINESPLINE refines the spline by Boehms algorithm
    t = knots;
    c = coefs;
    mu = k;
    while x >= t(mu+1)
        mu = mu + 1;
    end;
    c = [c c(end)];
    i = n;
    while i >= mu + 1
        c(i) = c(i-1);
        i = i - 1;
    end;
    i = mu;
    while i >= mu - d + 1 && i ~= 1
        alfa = (x - t(i)) / (t(i + d) - t(i));
        c(i) = (1 - alfa) * c(i-1) + alfa * c(i);
        i = i - 1;
    end;
    t = [t(1:mu) x t(mu+1:end)];
end
```

Ta algoritem je implementiran točno po definiciji. Najprej poiščemo tak mu, da bo približek x padel vmes med t_{mu} in t_{mu+1}. V teoretični razlagi ima mu isto vlogo kot črka l. Spremenljivko i nastavimo na n, ki predstavlja dolžino seznama koeficientov. Posvetimo se drugemu primeru iz (4.4) in vse koeficiente, ki so večji ali enaki od $mu + 1$ zamaknemo za eno mesto naprej. Potem se posvetimo tretjemu primeru iz (4.4) in popravimo kritične koeficiente. Prvega primera nam ni potrebno obravnavati, saj se koeficienti do mesta $mu - d$ ne spremenijo. Na koncu sledi še vstavljanje ničle poligona x v seznam vozlov.

Celoten postopek se ustavi, ko k preseže število koeficientov, torej smo padli ven iz intervala, na katerem iščemo ničle. V tem primeru gremo ven iz zanke in vrnemo rezultat. Ta je lahko seznam ničel polinoma, lahko pa je niz „*NO ZERO*“. Ta niz se pojavi, če nismo našli ničle po predpisanem številu iteracij, torej kadar zlepek nima ničel. Ponavadi že sam kontrolni poligon nima ničel in pridemo že kar v prvi iteraciji do konca, toda v splošnem to pravilo ne velja.

Zaključimo še z tremi primeri.

Primer: Vzemimo polinom $p(x) = x^4 + 6x^2 + 3x - 4$. To je polinom (2.1) iz prvega poglavja. Algoritem bomo pognali na intervalu $[-10, 10]$. Rezultat je prikazan na sliki 4.7.

```
>> p = @(x) x^4+6*x^2+3*x-4

p =

    @(x)x^4+6*x^2+3*x-4

>> findZeros(p, -10, 10)

ans =

   -1.0000    0.5918

fx >>
```

Slika 4.7: Rezultat algoritma za funkcijo $p(x) = x^4+6x^2+3x-4$ na intervalu $[-10, 10]$

Primer: Naj bo dan zlepek z vozli $\boldsymbol{t} = (0, 0, 0, 0, 1, 1, 1, 1)$ in zaporedjem koeficientov $\boldsymbol{c} = (-1, -1, \frac{1}{2}, 0)$. Rezultat je prikazan na sliki 4.8.

```
>> findZeros([0,0,0,0,1,1,1,1],[-1 -1 1/2 0])

ans =

    0.6961    1.0000

fx >>
```

Slika 4.8: Rezultat algoritma za zlepek dan z vozli $\boldsymbol{t} = (0, 0, 0, 0, 1, 1, 1, 1)$ in koeficienti $\boldsymbol{c} = (-1, -1, \frac{1}{2}, 0)$

Primer: Vzemimo sedaj polinom $p(x) = 10x^{10} - 2x^9 - 16x^8 + x^7 + x^6 + 4x^5 - 2x^4 + 5x^3 + 10x^2 + 25x + 13$ in naredimo enako kot prej. Rezultat je prikazan na sliki 4.9.

```
>> p = @(x)10*x^10-2*x^9-16*x^8+x^7+x^6+4*x^5-2*x^4+5*x^3+10*x^2+25*x+13

p =

    @(x)10*x^10-2*x^9-16*x^8+x^7+x^6+4*x^5-2*x^4+5*x^3+10*x^2+25*x+13

>> findZeros(p, -10, 10)

ans =

   -1.3228   -0.5890

fx >>
```

Slika 4.9: Rezultat algoritma za funkcijo $p(x) = 10x^{10} - 2x^9 - 16x^8 + x^7 + x^6 + 4x^5 - 2x^4 + 5x^3 + 10x^2 + 25x + 13$ na intervalu $[-10, 10]$

Opazimo lahko, da so dobljene ničle natančne samo do druge decimalke. Funkcijski vrednosti v teh dveh točkah sta 0.6535 in 0.0518. Ta odstopanja se pojavijo zaradi numeričnih napak, saj smo uporabili polinom visoke stopnje in iskali smo v okolici ničle.

Poglavje 5

Zaključek

V diplomskem delu smo spoznali popolnoma nov način iskanja ničel polinomov. Zgodbo smo pripeljali od začetka, torej definicije polinoma, do konca, torej implementacije.

Kot že omenjeno smo začeli z definicijo osnovnih pojmov. Spoznali smo polinome in ničle, pogledali pa smo si nekaj lastnosti, ki so za nas pomembne kot je naprimer osnovni izrek algebre 2.1 in njegovo posledico. Nič kaj posebno zanimivo poglavje za izkušenega bralca.

Naslednje poglavje pa je bistveno za razumevanje te metode. Interpolacija je že tako ali tako eden glavnih delov numerike v današnjem času in enako je s to metodo, saj temelji na njej. Prvi del poglavja je namenjen definiranju osnovnih pojmov za drugi in tretji del kot je na primer interpolacijski problem, pojem baznega polinoma in najpomembneje pojem deljenih diferenc, poskušamo pa tudi na čim lažji in čim bolj razumljiv način problem približati bralcu. V ta namen smo naredili nekoliko bolj obsežen in v podrobnosti opisan primer za deljene diference. Splošna interpolacija pa nam res služi samo kot ogrodje, za nas pa so bolj pomembne odsekoma polinomske funkcije, saj so B-zlepki poseben primer le-teh.

Srečamo še en pojem iz naslova diplomskega dela, to so vozli in še prej stične točke. Ta del je ključen za razumevanje algoritma. Na teh vrednostih se zlepek spremeni, lahko se tudi zlomi, če je nepovezan. Ponavadi pa

imamo opravke s povezanimi. Na te smo se namreč osredotočili v celotnem diplomskem delu. Preden smo šli na definiranje B-zlepkov smo spet naredili primer 3.1, ki pokaže obnašanje zlepka v stičnih točkah.

Tretji del interpolacije pa je nekoliko bolj matematičen, namenjen pa je izpeljavi bazičnih zlepkov oziroma B-zlepkov. Na koncu dobimo lepo rekurzivno formulo, ki je numerično absolutno stabilna, kar nam da zelo dober razlog za njeno uporabo.

V zadnjem delu dela smo šele povedali v podrobnosti kako ta algoritem deluje, saj smo potrebovali veliko teoretičnega materiala. Tudi na tem mestu nam je ostalo veliko za povedati, na primer kaj je to kontrolni poligon, kako se vstavlja vozle v zlepek, morali pa smo odgovoriti na vprašanje če vstavljanje vozla pokvari približek funkcije. Odgovor je seveda ne, saj koeficiente B-zlepkov priredimo po Boehm-ovem algoritmu 4.3. V tem poglavju bi se lahko še bolj poglobili v samo numerično stabilnost in v hitrost konvergence, toda to je tema za kdaj drugič, opisana pa je že v [5].

Sam algoritem je uporaben bolj za praktične primere uporabe. To pomeni, da pridejo v poštev samo polinomi stopnje do 10. Pri višjih stopnjah se numerična stabilnost zmanjša, čeprav je algoritem sam absolutno stabilen. Takoj ko imamo opravka z višjimi stopnjami, lahko polinomi zelo oscilirajo, pojavijo pa se tudi težave pri računanju vrednosti. Majhne spremembe v podatkih lahko povzročijo velike spremembe pri rezultatu, čemur pa se poskušamo izogniti.

Samo diplomsko delo je bolj kot ne opis dejanske metode, na koncu pa je implementiran program, katerega bi se še vedno dalo optimizirati. Poskrbeti bi morali namreč še za izračun koeficientov, omogočiti razbitje zlepka na dva dela, saj nam tudi to pride prav v kakšnih primerih, toda to je tema za kakšen drug dan.

Slike

Tabele

Algoritmi

Literatura

[1] Wolfgang Boehm. Inserting new knots into B-spline curves. *Computer-Aided Design*, 12(4), 1980.

[2] Michael A Jenkins in Joseph F Traub. A three-stage variable-shift iteration for polynomial zeros and its relation to generalized Rayleigh iteration. *Numerische Mathematik*, 14(3):252–263, 1970.

[3] Tomaž Košir. Osnovni izrek algebre, 2009. Dostopno na `http://www.fmf.uni-lj.si/~kosir/poucevanje/skripta/osnovniizrek.pdf`.

[4] Jernej Kozak. *Numerična analiza*. DMFA-založništvo, 2008.

[5] Knut Mørken in Martin Reimers. An unconditionally convergent method for computing zeros of splines and polynomials. *Mathematics of Computation*, 76(258):845–865, 2007. Dostopno na `http://www.ams.org/journals/mcom/2007-76-258/S0025-5718-07-01923-0/S0025-5718-07-01923-0.pdf`.